Université Paris-Saclay
M2 Droit de la création et du numérique

Approche de l'élaboration et du fonctionnement des logiciels

Alain Delaët

objectif des séances 2 & 3

faire l'apprentissage de la programmation avec le langage Python

Python nous offre

- des entiers, avec des opérations arithmétiques
- des variables pour mémoriser le résultat de calculs

```
age = 2048 - annee
```

- une instruction `print` et des chaînes de caractères

```
print("Vous aurez", age, "ans en 2048.")
```

- des instructions conditionnelles avec `if` / `elif` / `else`
- des boucles bornées (`for`) et tant que (`while`)
- du ≪ hasard ≫ avec `randint`

la boucle `for` de Python nous permet de parcourir les entiers de 0 inclus à 100 exclu, comme ceci

```
for i in range(100):
    print(i)
```

```
0
1
...
99
```

(100 lignes au total)

on peut passer un premier paramètre à `range` pour spécifier le nombre de départ

```
for i in range(20, 100):
```

```
20
21
...
99
```

(80 lignes au total)

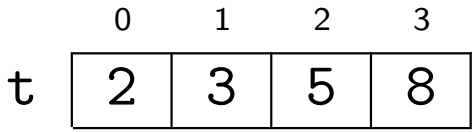
aujourd'hui

1. les tableaux
2. les fonctions

tableaux

si on a de très nombreuses données à manipuler, il peut être compliqué, voire impossible, de les ranger dans autant de variables
on va se servir de la mémoire pour organiser nos données d'une façon plus efficace que "une variable = une valeur"
la solution la plus simple consiste à utiliser **un tableau**

un **tableau** est un ensemble de cases mémoire, consécutives, auxquelles on accède par des **indices**
exemple : un tableau `t` de taille 4 contenant les entiers 2, 3, 5 et 8



(les indices sont indiqués au-dessus des cases)

le tableau rappelle fortement la description de la mémoire vive
(cours 1, transparents 19 et 20)
de fait, le tableau tire partie de l'accès direct à la mémoire
(RAM = *Random Access Memory*)

en Python, on peut créer un tableau en donnant ses éléments entre crochets

```
t = [2, 3, 5, 8]
```

on obtient la taille d'un tableau avec `len(t)`

```
4
```

(pour *length*)

on accède à la case i avec `t[i]`

```
5
```

on peut modifier le contenu d'une case avec une affectation, comme pour une variable

```
t[2] = 42
print(t)
[2, 3, 42, 8]
```

la pyramide des âges des français (ici au 1er janvier 2022) peut être donnée sous la forme d'un tableau

```
pa = [ 690942, 695063, 716123, ..., 19134, 31037 ]
```

où la case d'indice i contient le nombre de français ayant un âge entre i inclus et $i + 1$ exclu
on le voit bien, il aurait été extrêmement fastidieux d'introduire plus de 100 variables pour toutes ces valeurs

pour calculer le nombre de français ayant entre 10 et 20 ans, on peut faire

```
pa[10] + pa[11] + pa[12] + ... + pa[19]
```

mais c'est un peu fastidieux à écrire

on peut avantageusement utiliser une **boucle** pour faire ce calcul

```
n = 0
for i in range(10, 20):
    n = n + pa[i]
    print(n, "français entre 10 et 20 ans")
```

parcours de tout le tableau

pour parcourir tout le tableau, on peut écrire

```
pop = 0
for i in range(len(pa)):
    pop = pop + pa[i]
print(pop, "français au 1er janvier 2022")
```

(utiliser `len(pa)` nous dispense de connaître la taille du tableau)

mais on peut aussi écrire

```
pop = 0
for x in pa:
    pop = pop + x
print(pop, "français au 1er janvier 2022")
```

ici, la variable `x` prend successivement toutes les valeurs contenues dans le tableau

chercher le plus grand élément dans un tableau

```
vmax = pa[0]
for i in range(1, len(pa)):
    if pa[i] > vmax:
        vmax = pa[i]
print("plus grande valeur :", vmax)
```

fonctions

si on est amené à écrire plusieurs fois les **mêmes instructions** dans un programme, on peut avantageusement les regrouper dans **une fonction**

une fonction qui affiche une petite “boîte” :

```
def boite():
    print("+" + ("-" * 20) + "+")
    print("|" + (" " * 20) + "|")
    print("+" + ("-" * 20) + "+")
```

- le mot-clé `def` introduit la fonction, ici appelée `boite`
- le corps de la fonction est en retrait
- aucune instruction n’est exécutée

pour exécuter les instructions qui composent la fonction, on **appelle** la fonction, comme ceci

```
boite()
```

```
$ python fichier.py
+--------------------+
|                    |
+--------------------+
```

- on a **isolé** un concept (par ex. dessiner une boîte), et on l'a **nommé**
- on peut appeler la fonction **plusieurs fois** dans le programme
- on peut **changer** le comportement de la fonction, en n'intervenant qu'à un seul endroit

une fonction peut avoir des **paramètres**
exemple :

```
def boite(n):
    print("+" + ("-" * n) + "+")
    print("|" + (" " * n) + "|")
    print("+" + ("-" * n) + "+")
```

ici le paramètre `n` spécifie la taille de la boîte qui est dessinée

la valeur du paramètre est spécifiée à l'appel

```
    boite(30)
    boite(42)
```

```
+--------------------+
|                    |
+--------------------+
+----------------------------+
|                            |
+----------------------------+
```

on a déjà croisé une fonction avec paramètres : `print`
c'est une **fonction prédéfinie** de Python

une fonction peut également **renvoyer un résultat**
exemple :

```
def moyenne(a, b, c):
    s = a + b + c
    return s / 3
```

- l'instruction `return` permet de **renvoyer** une valeur, ici `s / 3`
- la fonction peut faire des calculs intermédiaires, ici dans la variable `s`

une fonction qui renvoie un résultat peut être utilisée dans un calcul
exemple :

```
print(moyenne(17, 18, 20))
print(max(12, moyenne(8, 11, 10)))
```

```
18.333333333333332
12
```

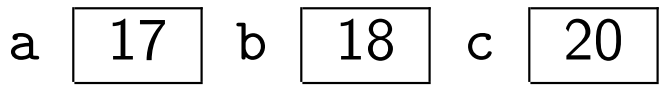
on a déjà croisé une fonction qui renvoie un résultat : `randint`

```
def moyenne(a, b, c):
    s = a + b + c
    return s / 3

    print(moyenne(17, 18, 20))
```

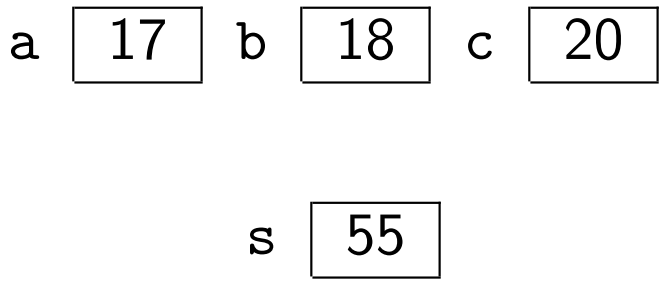
```
def moyenne(a, b, c):
    s = a + b + c
    return s / 3

    print(moyenne(17, 18, 20))
```



```
def moyenne(a, b, c):
    s = a + b + c
    return s / 3

    print(moyenne(17, 18, 20))
```



```
def moyenne(a, b, c):
    s = a + b + c
    return s / 3

    print(moyenne(17, 18, 20))
```

affiche `18.333333333333332`

une fonction qui ne renvoie pas de résultat — comme la fonction `boite` plus haut — est parfois appelée une **procédure**

afficher (`print`) ou renvoyer (`return`), ce n'est pas la même chose

- une fonction qui se contente d'afficher ne renvoie pas de valeur, et son appel ne peut être utilisé dans un calcul

```
print(1 + print(2))  // <- erreur
```

- une fonction qui renvoie une valeur n'affiche rien

```
moyenne(10, 13, 14)  // <- rien n'est affiché
```

une instruction `return` termine **immédiatement** l'exécution de la fonction exemple :

```
def bissextile(a):
    if a % 400 == 0: return True
    if a % 100 == 0: return False
    return a % 4 == 0
```

si l'année `a` est multiple de 400, alors la fonction renvoie `True` et ne teste pas si `a` est multiple de 100 ou de 4

à vous de jouer

écrire un programme qui calcule le nombre de jours depuis votre naissance

1. reprendre la fonction `bissextile` donnée plus haut
2. définir une fonction donnant le nombre de jours de l'année `a`

```
def nbjoursannee(a):
```

3. définir un tableau donnant le nombre de jours de chaque mois

```
joursmois = [0, 31, 28, ...]
```

(la case 0 n'est pas utilisée)

4. définir une fonction donnant le nombre de jours du mois `m` (1..12) de l'année `a`

```
def nbjoursmois(a, m):
```

5. définir une fonction qui calcule le nombre de jours depuis le 1er janvier 1970

```
def nbjours70(j, m, a):
```

6. calculer enfin le nombre de jours depuis votre naissance

on suggère fortement de tester au fur et à mesure
par exemple,

```
print(nbjoursannee(2016))
```

doit afficher 366 et

```
print(nbjoursmois(2016, 2))
```

doit afficher 29

- **vendredi 21 février**
  - la complexité du logiciel
-  prendre un jeu de carte